Pearls of Computer Science (202001022)

Pearl 000: Binary logic and computer architecture

Test of September 15, 2023

Answers

---

1. Binary numbers

6 pt (a) 110001

One can check that this is correct using weights of the positions, with the left-most bit having negative weight: $-1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^0 + 1 \cdot 2^0 = -15$.

5 pt (b) 26C

Conversion from binary to hexadecimal is easy in principle, as each group of 4 binary digits (bits) corresponds to 1 hexadecimal digit, starting from the right as that way the weights align ($2^4 = 16^1$).

6 pt (c) 680

Use the positional weights: $2 \cdot 16^2 + 10 \cdot 16^1 + 8 \cdot 16^0 = 680$ decimal.

4 pt (d) • With this system, we can represent all integers from 0 to 9, and some of them can be represented in multiple ways.

One way, albeit rather time-consuming, to verify this, is to simply try all 16 bit patterns from 0000 to 1111, and see what comes out.

A faster way is to first observe that the weights of the last three bits are the same as in 'normal' binary. So if the left-most bit is 0, the last three bits can represent the (decimal) numbers 0...7. Setting the left-most bit to 1 adds 2 to this, giving us 2...9. Considering both together, we see 0...9 are present, with 2...7 being represented in two different ways.

4 pt (e) • First invert all bits and then add the binary number 00001 to it.

Multiplying by -1 as the question asks, is the same as making the number negative. So the rule here is basically how you'd write down a negative number in 2-complement binary. Alternatively, you could verify this rule by applying it to a written-out table of 2-complement numbers.

Another way of seeing it is that, starting from a number n which is positive, so its left-most bit is 0, by inverting all the bits, you get $-2^5 + (2^4 + 2^3 + 2^2 + 2^1 + 2^0 - n)$: the first term is the effect of making the left-most bit 1 (as its weight is -2^5), and the others are the effect of flipping the other bits (if the ith bit in n is 1 it contributes $1 \cdot 2^i$ to n, which after flipping becomes $0 \cdot 2^i$, and vice versa). The result is $-1 - n$, and adding 1 to this makes it $-n$. A similar argument holds if we started out with a negative number.

Continued on next page...

2. Boolean logic

6 pt (a)

A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

10 pt (b) From top to bottom:
complement
identity
wrong
DeMorgan
distributive

6 pt (c) $\overline{A \cdot B} \cdot \overline{B + C}$

The above is the formula you get from directly writing down how the gates are connected.
If you want, it can be further simplified to $\overline{B + C}$ or (using DeMorgan) $\overline{B} \cdot \overline{C}$. (Note that this shows that the output does not depend on A, so this circuit is unnecessarily complicated.)

3 pt (d) • AND gate

This follows directly from DeMorgan's theorem.

Continued on next page...

12 pt **3. Problem 3**

One example of a correct solution:

	read address 1 / write address	read address 2	instruction	explanation
Timeslot 0	2	1	1	calculate $x \cdot y$ and store this in R2
Timeslot 1	3	1	1	calculate $x \cdot z$ and store this in R3
Timeslot 2	1	2	0	calculate R2+x, which equals $xy+x$, and store the result in R1
Timeslot 3	1	3	0	finally, add R3 (containing xz) also to R1

There are many other correct answers; for example interchanging the two multiplications, or interchanging the two additions. It can even be done in just three instructions, by first adding R2 and R3 (i.e., $y + z$) and then multiplying this by x.

The most frequently made error is prematurely overwriting a register whose (original) contents you still need further on. For example, if in the above solution the first instruction would be made to write to R1, it would overwrite the original contents of R1 (namely, x) which is still needed for the later steps.

Some students used the fourth register, and apparently assumed that the initial contents of that register would be 0. However, it was not given that this is the case, so such solutions are not correct. (Though of course you could still get partial points.)

Continued on next page...

4. Problem 4

18 pt (a)

R17	R18	R19	branch/comment
		1	
1			
2			
		2	
	2		
	1		
			jump to DEC R18
	0		
			jump not performed
	1		
1			
			jump to DEC R18
	0		
			jump not performed
0			
			jump not performed

Be careful about calculating to where the BRNE jumps. If there were no jump, it would continue with the first instruction after the BRNE, so e.g. –2 means it goes to 2 addresses earlier, which in this case is the DEC R18 instruction.
Another way of remembering this is that we often finished the Arduino programs with RJMP –1, which gives an infinite loop. So jump –1 goes to the jump instruction itself, so jump –2 must go to the instruction just before that.

Some students combined multiple instructions on a single line, e.g. both initial LDIs. The exam clearly stated that one line should be used per instruction, though we didn't subtract much for not doing so. However, it makes it harder to follow what is going on, and it makes answering 4b (counting the number of clock cycles) harder.

Furthermore, note that BRNE is also an instruction, so it should also get a line of its own, even if the branch is not taken because the condition is not satisfied, and even though none of the register values change. It also takes (at least) one clock cycle.

The MOV instruction copies the contents from R19 to R18, without changing the contents of R19. In fact, the explanation in the problem points out that despite the mnemonic 'MOVe', it actually copies. Some students seem to assume that the contents of R19 become 0, or that R19 becomes empty, after the MOV. A register can't really be 'empty': it's just a memory for 8 bits, each of which is either 0 or 1, but nothing else. (Of course, it may be that the content of the register is not known to us, like the values of R18 and R19 in the first few instructions. But that doesn't mean the registers are empty.)

5 pt (b) 19 cycles.
Explanation: 16 instructions are executed, as can be seen in the table. 2 of these instructions are BRNEs that do not jump so they take 1 cycle just like the 'normal' instructions, and 3 are BRNEs that do jump and thus each take 2 cycles rather than 1. Hence a total of 16 + 3 = 19 cycles.

For grading this question, we checked whether your answer matched what you wrote in question (a).

Continued on next page...

15 pt **5. Problem 5**

This calculates $\lfloor X/Y \rfloor$, i.e., X/Y rounded down to an integer.
Using Python's // notation for integer division, this is $X//Y$.

(But note that the // notation is far from universal; in fact, in many programming languages // starts a comment, while integer division is denoted by the single / . The $\lfloor$ and $\rfloor$ are the standard mathematical symbols for rounding down. On the exam, we accepted everything that clearly indicated the rounding, even plain words.)

Looking at the code, we see it repeatedly subtracts R18 from R17, until the result is negative. R18 contains Y all the time (as there's nothing which changes R18), while R17 initially contains X. So we subtract so many times Y from X that the result in R17 is negative.

While doing those subtractions, R19 gets incremented, so it counts how many subtractions are done. This looks like a division: how many times can Y be subtracted from X? So the result in R19 is something like X/Y.

To be precise, we need to be careful about exactly how R19 is initialized, and note that it is decremented by 1 at the end, and carefully consider exactly when the loop is terminated. Due to these details, there might easily be e.g. a +1 or -1 correction to that X/Y expression. Furthermore, we need to think about rounding, as X/Y may not be an integer while R19 definitely is. The easiest way to solve this is to try a few examples; then one sees that the final result is X/Y rounded down.

For example, if $X = 2Y$, the subtraction will be done 3 times (because after 2 subtractions, R17 is 0, which does not yet set the carry flag). R19 starts at 0, gets incremented 3 times (once for each subtraction of R17), and at the end gets decremented by 1, resulting in 2. So indeed, R19 equals X/Y, without a further correction.

And if e.g. $Y = 1.5\,X$, then already after two subtractions R17 would be negative, resulting (after the DEC R19) in R19 being 1 at the end of the program; so the 1.5 is rounded down.